TECNOLOGICO NACIONAL DE MEXICO

INSTITUTO TECNOLOGICO DE ORIZABA

Asignatura: Estructura de Datos

Carrera: Ingeniería en Informática

Estudiantes:

- Hernandez Angón Diego - No. Control 21010193
- Dorantes Rodríguez Diego Yael- No. Control 21010184

Profesora: María Jacinta Martínez Castillo

Grupo: 3a3A

Fecha de entrega: 22/04/2023

Reporte de unidad 1: Introducción a las estructuras de datos

INTRODUCCION

En el siguiente reporte, abordaremos el tema de los tipos de datos abstractos, utilizando recursos como un IDE de programación, además de adquirir conocimiento acerca de los TDA, obteniendo como resultado un proyecto enfocado a los datos ordenados y datos desordenados. Aprenderemos a diferencias las estructuras e datos tanto lineales como las no lineales

COMPETENCIA ESPECIFICA

Desarrollar la habilidad de poder reconocer e identificar los tipos de datos abstractos, mediante la utilización de estructuras de datos lineales y no lineales, además de poder trabajar correctamente con datos desordenados y datos ordenados mediante código de programación

MARCO TEORICO

Las estructuras de datos son una manera específica de ordenar y organizar la información en una computadora, con el fin de hacer un uso más efectivo de ella. Existen distintos tipos de estructuras de datos, cada uno apropiado para diferentes propósitos, incluso algunas están diseñadas especialmente para realizar tareas muy particulares. Estas estructuras de datos son una herramienta importante para manejar grandes volúmenes de información de manera efectiva, en aplicaciones como bases de datos extensas o servicios en la web.

Un Tipo de Dato Abstracto (TDA) es un modelo que define valores y las operaciones que se pueden realizan sobre ellos. Y se denomina abstracto ya que la intención es que quien lo utiliza, no necesita conocer los detalles de la representación interna o bien el cómo están implementadas las operaciones.

Es por esto una práctica que nos provee un grado de abstracción que permite desacoplar al código que usa un TDA de aquel código que lo implementa.

Ejemplo:

```
tipo Pantalla
operacion escribir (Pantalla, String)
operacion leer (Pantalla) -> String
```

Se define un tipo Pantalla y dos operaciones sobre él.

La operación tiene un nombre y, como una función, un dominio e imagen. Se lee así

```
operacion nombre (parametro1,
parametro2, ...) -> retorno
```

En nuestro caso simple, tenemos dos funciones: para escribir en pantalla, y para leer.

Fíjense que, mediante este tipo, yo puedo escribir código que opere con la pantalla, sin conocer cómo realmente se escribe o lee de ésta.

- Memoria dinámica

  Es memoria que se reserva en tiempo de ejecución. Su principal ventaja frente a la estática, es que su tamaño puede variar durante la ejecución del programa. (En C, el programador es encargado de liberar esta memoria cuando no la utilice más). El uso de memoria dinámica es necesario cuando a priori no conocemos el número de datos/elementos a tratar.
- Memoria estática

  Es el espacio en memoria que se crea al declarar variables de cualquier tipo de dato (primitivas [int, char] o derivados [struct, matrices, punteros]). La memoria que estas variables ocupan no puede cambiarse durante la ejecución y tampoco puede ser liberada manualmente

MATERIAL Y EQUIPO

El proyecto se lleva a cabo con los debidos requisitos necesarios. Contar con una computadora con los componentes adecuados, capaz de soportar IDES de programación con lenguaje de java. Se necesita de software específico para la creación de código en Java, el cual puede ser Eclipse o NetBeans. Se cuenta

además con los conocimientos otorgados por la profesora en clase, además de material de apoyo para reforzar el conocimiento

DESARROLLO Y RESULTADOS

Para el desarrollo de los códigos para datos ordenados y datos desordenados utilizamos el equipo antes necesario, además de contar con una plantilla creada anteriormente con métodos para agregar e imprimir datos para mayor comodidad y eficiencia de los códigos.

- Datos desordenados

Comenzamos creando una clase para los métodos del programa

```
package Actualizacion;
import EntradaSalida.tools;

public class DatoSimple {

        private Object datos[];
        private byte p;

        public DatoSimple(byte tam)
        {
            datos=new Object[tam];
            p=-1;
             }
```

Utilizaremos métodos para comprobar si el arreglo que vamos a llenar este vacío, también utilizaremos un método para imprimir y agregar datos desordenados en nuestro programa

```
        public boolean validaVacio()
        {
           return (p==-1);
        }
        public void almacenarDato()
        {
            if(p<datos.length)
            {
                datos[p+1]= tools.LeerEntero("Escribe un numero");
                p++;
            }
            else tools.ImprimeErrorMsje("Array lleno");

        }
        public  String imprimeDatos()
        {
            String cad="";
            for (int i = 0; i <=p; i++) {
                cad+=i+"["+datos[i]+"]"+"\n";

           }
           return ("\n"+cad);
        }
```

Por último, se agrega un método para la búsqueda, esta debe ser secuencial, ya que los datos están desordenados, por lo que el siguiente método nos queda asi:

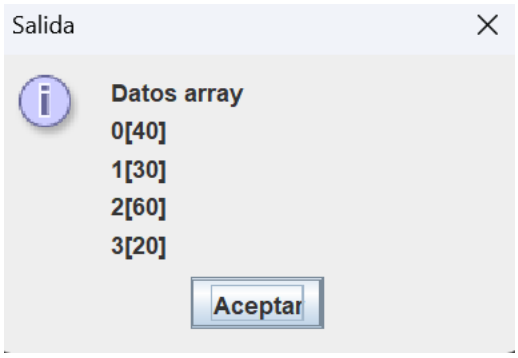
```
public byte buscarSecuencial(Object dat) {
    byte i=0;
    while(i<=p && !dat.equals(datos[i]))
        i++;
    return (dat.equals(datos[i]))?i:-1;
}
public void eliminaDato(byte pos) {
    for(int j = pos; j <= p; j++) {
        datos[j] = datos[j+1];

    }
    p--;
}
```

Crearemos un menú para todos los métodos de los datos desordenados, similar al que utilizaremos con los datos ordenados:

```
DatoSimple obj = new DatoSimple((byte)10);
String sel="";
do {
    sel=tools.boton(menu);
    switch(sel) {
    case "Agregar":
        obj.almacenarDato();
        break;
    case "Imprimir":
        tools.ImprimeMsje("Datos array" + obj.imprimeDatos());
        break;
    case "Buscar":
        if(obj.validaVacio())
            tools.ImprimeErrorMsje("Array vacio");
        else {
            //pos = obj.buscarOrdenada(arreglo,tools.LeerEntero("Ingrese el valor"));
            byte pos=obj.buscarSecuencial(tools.LeerString("Nombre a buscar"));
            if(pos!=-2)
                tools.ImprimeMsje("Se encuentra en la posicion " + pos);

            else tools.ImprimeErrorMsje("Dato no encontrado");
        }
        break;
```

El resultado será un menú que nos permita agregar datos desordenados de esta forma:

Los datos quedan de esta forma:



- Datos ordenados

Para los datos desordenados, ocuparemos métodos similares pero que tienen sus diferencias, diseñados para los datos ordenados:

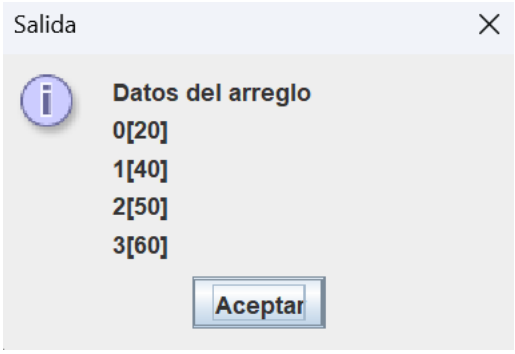
```
package Actualizacion;
import EntradaSalida.tools;

public class DatosOrdenados{
private int ordenados[];//Definir en privado el ar

private byte p; // Es un sub indice, equivalente a

//Despues se pone constructor que recibe tamaño pa
public DatosOrdenados(int tam)//Un constructor no

{ //Todo constructor debe tener llave de inicio y
   ordenados = new int[tam];// Se crea el arreglo
   p=-1;//Subindice, -1 para validar si esta vacio
}

public boolean validaVacio()
{
return(p==-1);
}

public String imprimeDatos()
{
  String cad="";
  for(int i=0;i<=p;i++){
     cad+=i+"["+ordenados[i]+"]"+"\n";//Se utiliza
    }
   return "\n"+cad;
}
public byte busSecuencialOrdenada( int dat) //Esta
  {
    byte i=0;
    while(i<=p && ordenados[i]<dat)
    i++;
    return (byte)((i>p||ordenados[i]>dat)?-i:i);//
```

El menú también es similar al de datos desordenados, con unas pequeñas modificaciones en cuanto a los nombres de los métodos, pero con la misma estructura:

```
String sel="";
do
{
    sel=boton(menu);
    switch(sel)
    {
        case "Agregar":
            obj.insertandoOrdenado();
        break;
        case "Imprimir":
            if(obj.validaVacio())
                tools.ImprimeMsje("Arreglo vacio");
            else
                tools.ImprimeMsje("Datos del arreglo" + obj.imprimeDatos());
        break;
        case "Busqueda":
            if(obj.validaVacio())
                tools.ImprimeErrorMsje("Arreglo vacio");
            else
            {
                pos = obj.busSecuencialOrdenada(tools.LeerEntero("Ingrese el numero"));
                if (pos >= 0)
                    tools.ImprimeMsje("Se encuentra en la posicion: " + pos);
                else
                    tools.ImprimeMsje("Se debe de insertar en la posicion" + pos * (-1));
            }
        break;
```

RESULTADO

El resultado será, ahora con datos ordenados automáticamente por el programa, como se ve a continuación:



CONCLUSIONES

Hemos llegado a la conclusión de que, utilizando los TDA podemos ser capaces de crear códigos para datos ordenados y desordenados, y algunas de las estructuras de datos más comunes como lo son matrices, listas enlazadas, pilas, colas, árboles y grafos. La comprensión de las estructuras de datos es un conocimiento esencial para cualquier programador, y su aplicación adecuada puede mejorar significativamente la eficiencia y la capacidad de manejo de los datos de un programa.

BIBLIOGRAFIA

- Anónimo. S.F. Tipos de datos abstractos. EJEMPLOS. https://ejemplos.net/tipos-de-datos-abstractos/
- Anónimo. S.f. Tipo de dato abstracto. EcuRed. https://www.ecured.cu/Tipo_de_Dato_Abstracto
- Krypton Solid. S.f. ¿Qué son las estructuras de datos? – Definición de Krypton Solid. https://kryptonsolid.com/que-son-las-estructuras-de-datos-definicion-de-krypton-solid/
- Katya P. S.f. Introducción a las Estructuras de Datos. https://virtual.usalesiana.edu.bo/web/conte/archivos/280.pdf
- Rafael Camps. S.f. Introducción a las bases de datos. https://web.seducoahuila.gob.mx/biblioweb/upload/P06_M2109_02147.pdf